Beyond Traditional Tuning: Unveiling Metaheuristic Operator Trends in PID Control Tuning for Automatic Voltage Regulation

Daniel F. Zambrano-Gutierrez, Jorge M. Cruz-Duarte, José Carlos Ortiz-Bayliss, Iván Amaya
School of Engineering and Sciences
Tecnologico de Monterrey, Monterrey 64849, Mexico
E-mails: {A00836756, jorge.cruz, jcobayliss@tec.mx, iamaya2@tec.mx }@tec.mx

Juan Gabriel Avina-Cervantes
Telematics Group, Department of Electronics Engineering
University of Guanajuato, Salamanca 36885, Mexico
E-mail: avina@ugto.mx

***Abstract*—Effective optimization of model variables is essential yet demanding in engineering and industrial processes. Metaheuristics (MHs) offer a proficient approach, but their design and tuning incorporate notable challenges. Automated Algorithm Design (AAD) methodologies provide a solution by enabling automated algorithm construction. This study utilizes a Hyper-Heuristic (HH) framework within AAD, which obtains a tailored MH to optimize a Proportional, Integral, and Derivative (PID) controller in an Automatic Voltage Regulation (AVR) system. We identify a preference for search operators from Spiral Dynamic, Swarm Dynamic, and Differential Mutation families, offering valuable insights for MH algorithm design in complex electrical systems. Our contributions include a novel methodology for distilling search operators for specific problem families and presenting effective search operators for MHs in electrical engineering scenarios. The study highlights the importance of precise controller tuning, demonstrated through the effectiveness of the tailored MH compared to others.**

***Index Terms*—Automated Design; Hyper-heuristic; Metaheuristic; Control Engineering; Parameter Tuning; Complex Systems.**

I. Introduction

OPTIMIZATION methods, particularly Metaheuristics (MHs), play a crucial role in solving practical engineering problems [1], [2]. With the advent of high-performance computing, the focus has shifted towards developing heuristic algorithms capable of delivering reasonable solutions efficiently for complex problems, such as non-differentiable multi-modal functions in continuous optimization and NP-hard problems in discrete optimization [3], [4]. Indeed, while resource constraints are often manageable in many practical scenarios today, the reliance on sub-optimal optimization methods is evidently an intriguing paradox.

The increasing complexity of these problems has been driven by the availability of high-performance computers, which has oriented research toward creating heuristic algorithms that prioritize obtaining good solutions in a reasonable time. In this context, the need to innovate more efficient solving methods has encouraged the exploration of alternative design approaches that overcome manual design limitations, which are generally slow, biased, and error-prone. In addition, the literature presents a wide range of well-studied MHs. Some of them exhibit a solid mathematical, physical, and algorithmic formulation, e.g., Genetic Algorithms [5], Simulated Annealing [6], and Particle Swarm Optimization [7]. Others, in contrast, have been deemed as 'novel' by certain members of the metaheuristic community and are based on well-elaborated metaphors [8]. These are just old methods in new packaging [9]–[11].

To address this redundancy, a modular structure for configuring or analyzing custom MHs through a metaphor-free description has been proposed, offering various advantages [12], [13]. With this modular approach, an efficient way to obtain the suitable MH for a given problem is by automatically and intelligently generating MHs through a Hyper-Heuristic (HH) model [14]. This approach established the way for the automatic and intelligent generation of MHs using HHs [14], a high-level technique in Automated Algorithm Design (AAD) predominantly applied in combinatorial optimization but increasingly relevant in continuous problems [15], [16]. Besides, this framework has opened up many opportunities for MHs to solve engineering tuning complex problems, such as those related to electrical and robotic systems [17], [18].

Despite the extensive exploration of MHs in various engineering problems, a gap remains in the literature: identifying specific search operators within MHs that are most effective for particular engineering challenges. The performance of MHs is well-known to be closely linked to the characteristics or landscape features of the problem at hand [19]. Thus, we aim to identify those search operators that best suit a particular problem. This work makes three significant contributions to the field, such as:

1) It introduces a methodology for distilling the search operator collection for a given problem family.
2) It presents a collection of well-performing search operators tailored explicitly for metaheuristics in an

electrical engineering practical scenario.

3) It provides a thorough analysis of the influence of specific components in the development of metaheuristics, offering insights into how they affect overall performance.

As the particular case study, we consider the fine-tuning problem of Proportional, Integral, and Derivative (PID) controllers for Automatic Voltage Regulation (AVR) systems [20]. These controllers require proper and accurate tuning for the correct operation of the AVR system [21], [22]. Moreover, we utilize the HH framework detailed in [23] to custom-tune the PID controller and to identify the most effective MH operators for this specific application.

II. Theoretical Foundations

This section describes essential concepts related to automated algorithm design, particularly for metaheuristics.

A. Optimization

An optimization process can be represented as follows: consider a search space $\mathfrak{S} \in \mathbb{R}^n$ and a feasible subset $\mathfrak{M} \subseteq \mathfrak{S}$ representing all solutions satisfying the problem constraints. Let f: $\mathfrak{S} \to \mathbb{R}$ be an objective function that assigns a real value to each element in $\mathfrak{S}$. The optimization process is then defined as finding the element $\mathfrak{s}_* \in \mathfrak{S}$ such that $f(\mathfrak{s}_*)$ is optimal concerning all other elements in $\mathfrak{S}$. Formally, the optimization problem can be expressed as

$$\mathfrak{s}_* = \underset{\mathfrak{s}\in\mathfrak{S}}{\operatorname{argmin}} \{f(\mathfrak{s})\} \quad \text{or} \quad \mathfrak{s}_* = \underset{\mathfrak{s}\in\mathfrak{S}}{\operatorname{argmax}} \{f(\mathfrak{s})\}, \tag{1}$$

where the choice of argmin or argmax depends on whether the objective is to minimize or maximize the function f [24].

B. Heuristics

In its most basic form, a heuristic $\vec{h} \in \mathfrak{H}^{\varpi}$ is a procedure or operator that creates or modifies a solution for a given problem instance [25]. Thus, we formally represent a heuristic as a sequence of operations $\vec{h}=(h_2,\ldots,h_k)$, where each operation is an action that transforms a candidate solution. These operations can be classified into two levels of abstraction:

Low-Level Heuristics: These strategies interact directly with the problem domain to generate a candidate solution from scratch or modify a current candidate solution.

High-Level Heuristics: These strategies do not modify individual solutions directly but operate by selecting or combining low-level heuristics within the heuristic space.

In the remaining subsections, we briefly describe three particular cases of heuristics.

1) Simple Heuristics (SHs): These are strategies based on basic heuristics that interact directly with the problem domain or heuristic space $\mathcal{H}$. Besides, SHs represented by $h \in \mathcal{H}$ can operate at a low or high level of abstraction, producing or modifying a candidate solution $\mathfrak{s} \in \mathfrak{S}$. SHs can commonly be classified into three categories [12], [26]:

Constructor h_c: An SH generates new candidate solutions from an empty initial condition, i.e., $\mathfrak{s} \leftarrow h_c(\emptyset)$.

Perturbator h_p: An SH modifies one or more existing candidate solutions $\mathfrak{s}$, resulting in an updated solution $\mathfrak{s}'$, such that $\mathfrak{s}' \leftarrow h_p(\mathfrak{s})$.

Collector h_{co}: A function selects a candidate solution $\mathfrak{s} \in \mathfrak{S}$ and maps it to another candidate solution $\mathfrak{s}' \in \mathfrak{S}$ based on a selection criterion $c_s : \mathfrak{S} \to \mathbb{Z}_2$.

There are many ways to perturb and collect a candidate solution; some chosen from [12] are presented next.

Selector: Denoted as $h_s \in \mathcal{H}$, this heuristic decides whether to accept a new candidate solution $\mathfrak{s}'$ or retain the current $\mathfrak{s}$ based on a criterion $c_s : \mathfrak{S} \to \mathbb{Z}_2$. Common selectors like Greedy and Metropolis operate at a low abstraction level ($\mathfrak{s} = \mathbf{x}$, $\mathfrak{S} = \mathfrak{X}$) and are efficient in choosing solutions after applying perturbative heuristics [14].

Finalizer: Represented as $h_f \in \mathcal{H}$, this high-level operator concludes the heuristic search. It decides to either maintain the current solution $\mathfrak{s}$ or apply a transformation via a composite heuristic h_o, based on a completion criterion c_f that assesses solution quality, repetitiveness, and processing times.

Search Operator: The Search Operator (SO), h_o, is a composite of two simple heuristics: a Perturbator h_p followed by a selector h_s. It operates at multiple abstraction levels, transforming a candidate solution $\mathfrak{s}$ into a new one $\mathfrak{s}'$,

$$\mathfrak{s}' \leftarrow h_o(\mathfrak{s}) \equiv (h_s \circ h_p)(\mathfrak{s}) \tag{2}$$

SOs are fundamental in developing new MH, providing a systematic mechanism to design, analyze, and explore MH from a mathematical perspective.

2) Metaheuristics (MHs): These are commonly defined as a set of heuristics that operate under a structured scheme [27]. This method comprises a sequence of SOs acting in coordination until reaching a specific termination, known as the standard metaheuristic model [27], defined as

$$\mathrm{MH}\{\mathfrak{S}\} \equiv h_f(h_o) \circ h_c\{\mathfrak{S}\}, \tag{3}$$

where h_o is composed of a sequence of SOs, i.e., $h_o = h_{\varpi} \circ h_{\varpi-1} \circ \cdots \circ h_1, \forall h_k$, and h_c corresponds to a collector implemented as an initializer of the candidate solution. Plus, $\varpi \in \mathbb{Z}_{++}$ is the number of SOs in the MH.

3) Hyper-Heuristics (HHs): These are high-level algorithms that guide low-level heuristics to construct an efficient strategy that addresses an optimization problem $(\mathfrak{X}, f)$ [28]. The HH determines a sequence of heuristic actions to approximate the optimal solution instead of directly intervening in the problem solutions. The performance of a heuristic sequence h is computed by a quality function $Q(h; \mathfrak{X}, f) \to \mathbb{R}_+$, assigning a value that reflects the efficiency of $h \in \mathfrak{H} \subset \mathbb{H}^{\varpi}$ applied to $(\mathfrak{X}, f)$.

C. Automated Algorithm Design (AAD)

It has emerged as a significant field, concentrating on high-level algorithm selection and generation techniques

to solve problem families. Various approaches in AAD, including hyper-parameter optimization, modular algorithm construction, and technique integration, have been explored [23], [29]. Our study applies AAD in MHs for continuous optimization, addressing the Metaheuristic Composition Optimization Problem (MCOP) [30]. It involves optimizing the sequence of search operators and hyper-parameter configurations for MHs [31]–[33]. AAD utilizes a defined design space, encompassing possible MH configurations, algorithmic parameters, and an automatic configuration master algorithm. This comprehensive approach allows the systematic exploration of the design space to identify efficient configurations, employing either HHs or more sophisticated methods as high-level solvers for MCOP, as formulated:

$$(\mathrm{MH}_*; \mathbf{x}_*) = \underset{\mathrm{MH}\in\mathfrak{H},\, \mathbf{x}\in\mathfrak{X}}{\operatorname{argmax}} \{Q(\mathrm{MH}\,|\mathfrak{X})\}. \tag{4}$$

III. Voltage Regulators and Controllers

This section briefly describes the components of the electrical engineering scenario considered as the case study. The first one corresponds to the Automatic Voltage Regulation System (AVR) is a key component in the electric power generation and distribution infrastructure [34]. The AVR plays a crucial role in stabilizing electrical voltage by adjusting to input variations, load changes, and other external factors [35]. As depicted in Fig. 1(a), the AVR system comprises four interconnected subsystems, beginning with the sensing stage (G_s), essential for measuring the current-voltage and setting the stage for corrective actions. Plus, the error quantification stage compares the actual voltage (V_o) with the desired voltage ($\mathrm{V}_{\mathrm{ref}}$), followed by the Amplifier block (G_a), which amplifies the error signal. The Exciter (G_e) adjusts the Generator's magnetic field, while the Generator (G_g) converts mechanical to electrical energy. Fig. 1(b) shows the implemented model for obtaining the AVR's transfer function. Each subsystem, represented by their transfer function, $G_k(s) = K_k/(1 + s\tau_k)$, $\forall\, k = \{a, e, g, s\}$, plays a specific role in voltage regulation [36]. TABLE I lists these parameters chosen chosen based on [22].

TABLE I
Parameters for the AVR subsystems' transfer functions.

Subsystem	**Amplifier**	**Exciter**	**Generator**	**Sensor**
Gain	$\mathrm{K}_a = 10.0$	$\mathrm{K}_e = 1.0$	$\mathrm{K}_g = 1.0$	$\mathrm{K}_s = 1.0$
Time Constant	$\tau_a = 0.1$ s	$\tau_e = 0.4$ s	$\tau_g = 1.0$ s	$\tau_s = 0.01$ s

Now, without a controller, the AVR system displays undesirable behaviors such as high overshoot and steady-state errors (Fig. 2). These include a high overshoot (M_p = 67.42%), a long settling (T_s = 6.971 s) and rising (T_r = 0.754 s) times, and a steady state error (E_{ss} = 0.090 p.u.). Hence, we must implement a Proportional, Integral, and Derivative (PID) controller to guide the system towards a desired behavior. As its name indicates, this controller combines three fundamental control actions in response to the error signal, which compares

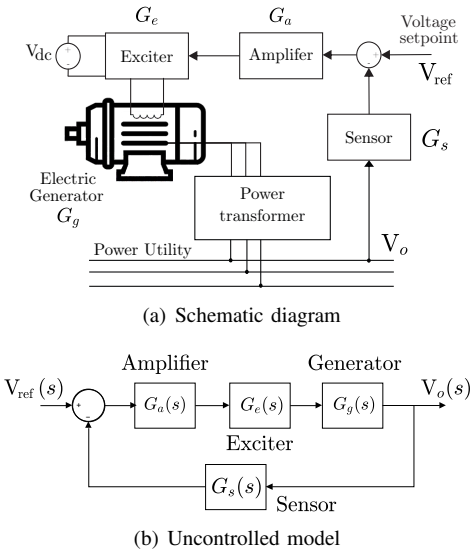

Fig. 1. Automatic Voltage Regulation System.

the reference and the sensor's output [37]. The PID's transfer function is given by $G_{\mathrm{PID}}(s) = \mathrm{K}_p + \mathrm{K}_i s^{-1} + \mathrm{K}_d\, s$, where K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively. Optimal tuning of these gains is vital for efficient voltage regulation, involving sophisticated calibration and possibly optimization algorithms.

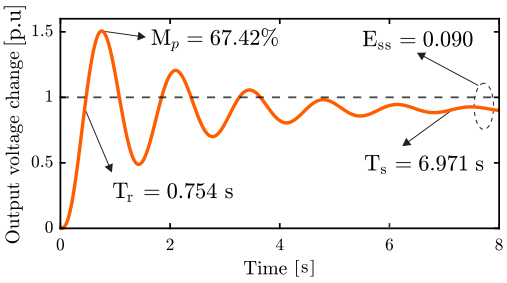

Fig. 2. Output voltage of the AVR system without a controller.

IV. Methodology

All experiments in this study were conducted using Python version 3.9 on an ASUS TUF Gaming F17, equipped with an AMD Ryzen™ 7 Processor 5700G, featuring 8 CPU Cores, and supported by 16GB RAM, running on a 64-bit Windows 11 operating system. The HH search was implemented using the CUSTOMHyS framework, which employs Simulated Annealing-based HHs to customize MHs to solve continuous optimization problems [23]. This framework is available at CUSTOMHyS PyPI. Fig. 3 illustrates the methodology employed for deriving optimal MHs specific to the case study involving an AVR system. In this context, the lower-level optimization problem is defined within a three-dimensional domain, utilizing the design vector $(\mathrm{K}_p, \mathrm{K}_i, \mathrm{K}_d)^\intercal \in \mathfrak{X} \subseteq \mathbb{R}^3$. Here, K_p ranges from 0.01 to 1.5, K_i spans from 0.01 to 1.5, and K_d varies between 0 and 0.5, corresponding to the parameters of the PID controller.

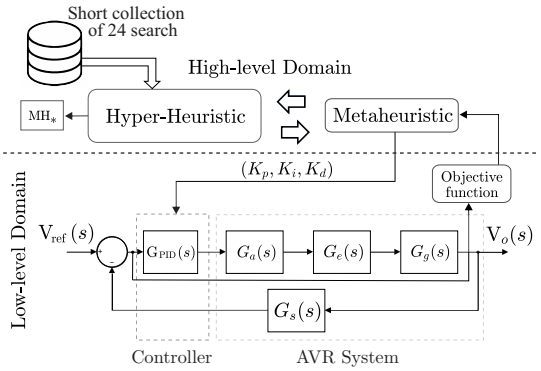

Fig. 3. Hyper-heuristic methodology implemented to generate optimal MHs for tuning a PID controller for an AVR system.

Our methodology is divided into two levels of abstraction: a low-level for the electrical power scenario and a high-level domain for the metaheuristic composition problem.

A. Low-Level Domain

In the literature, it is easy to identify multiple methodologies for the evaluation of the performance of controllers once integrated into dynamic systems [21], [22]. However, specific established performance metrics acquire particular relevance in the tuning process of such controllers. This work employs the Integral Time Absolute Error (ITAE) metric as part of our cost function. The ITAE is a performance index used in control theory to evaluate and improve system quality. Its formula integrates the time-weighted absolute error, highlighting errors that persist over time. A system optimized with ITAE minimizes overshoot and oscillations, achieving a smooth transient response. This metric is defined as

$$\text{ITAE} = \int_0^T \tau |e(\tau)| d\tau. \tag{5}$$

Despite the relevance of traditional metrics, other features are critically important in evaluating control system performance, such as Settling Time (T_s), Rise Time (T_r), Overshoot (M_p), and Steady-state Error (E_{ss}). These features indicate how quickly a system responds to changes and how accurately it maintains its output once it has reached a steady state. So, we designed the cost function to comprise these features, such as,

$$\text{F}_{obj} = \alpha \times (\text{M}_p + E_{ss}) + (1-\alpha) \times (\text{T}_\text{s} - t_r) + \gamma \times \text{ITAE}, \tag{6}$$

since $\alpha = 0.3$ and $\gamma = 0.1$ were determined for this study after different tests were performed. ITAE can reduce overshoot and dampen system oscillations. Lowering ITAE can decrease energy consumption by the control device or actuator. For example, optimizing the ITAE index in generating Pulse Width Modulation (PWM) signals can result in more energy-efficient control signals. This approach meets performance requirements and is essential in real-world applications.

B. High-Level Domain

For this purpose, we employ Simulated Annealing (SA), widely used as a high-level solver for HH applications [23]. Moreover, the performance of an arbitrary candidate sequence of search operators h_o to build a MH $= \langle h_i, h_o, h_f \rangle$ is determined by

$$\text{Q}(\text{MH} \,|\mathfrak{X}) = -(\text{med}(F_h) + \text{iqr}(F_h)), \tag{7}$$

with $F_h = \{f(\mathbf{x}_{r,*}) \mid \forall\ \mathbf{x}_{r,*} \in X_*\}$, since med and iqr are the median and inter-quartile range operators applied to the values from the fitness function $f(\mathbf{x}_{r,*})$, achieved after implementing the MH on N_r independent runs, i.e., $X_* = \{\mathbf{x}_{1,*}, \ldots, \mathbf{x}_{N_r,*}\}$.

We define the feasible heuristic space $\mathfrak{H} \in \mathbb{H}^{\varpi}$ in the high-level heuristic domain as displayed in TABLE II. This space comprises a set of SOs, each briefly listed in TABLE III. Each operator h_o is characterized by a perturbator h_p, a set of tuning parameters Ψ, and a predetermined selector h_s. From a programming perspective, we implemented a search operator as a 3-element tuple with these components, i.e., $h_o := (h_p, \Psi, h_s)$. More in detail, perturbators encompass strategies like differential crossover, differential mutation, and genetic mutation, which induce dynamic and stochastic behaviors inspired by natural phenomena and physical processes. Their tuning parameters include variables such as crossover rate ρ_{cr}, randomization count η_{rand}, pairing rate μ_{pool}, mutation rate μ_{rate}, selection coefficients σ_{sel}, swarm constants α_{swarm}, and random flight parameters β_{flight}, among others. The primary selectors used are the direct selection ('all') and the Greedy mechanism ('greedy').

TABLE II
COLLECTION OF SEARCH OPERATORS EMPLOYED AS THE HEURISTIC SPACE. THESE ARE COMPOSED OF A PERTURBATOR, ITS TUNING PARAMETERS, AND A DEFAULT SELECTOR, WHICH CAN BE CHANGED.

Perturbator	Tuning parameters	Selector
differential_crossover	crossover_rate	greedy
differential_mutation	num_rands, factor	all
genetic_crossover	mating_pool_factor	all
genetic_mutation	scale, elite_rate, mutation_rate	greedy
swarm_dynamic	factor, self_conf, swarm_con	all
firefly_dynamic	alpha, beta, gamma	all
spiral_dynamic	radius, angle, sigma	all
random_flight	scale, beta	greedy
random_sample	-	all
random_search	scale	greedy
local_random_walk	probability, scale	greedy
central_force_dynamic	gravity, alpha, beta, dt	all
gravitational_search	gravity, alpha	all

From the search operators shown in TABLE II, it has been decided to implement a short collection composed of 24 search operators described in detail in TABLE III. This table shows that two variants are available for each family of operators. This elaborate selection aims to balance and ensure a homogeneous probability distribution in selecting operators belonging to each family.

Lastly, to implement the HH procedure, each MH used a population size of 15 individuals and a maximum of 20

TABLE III
SHORT COLLECTION OF THE 24 SEARCH OPERATORS EMPLOYED.

Ids.	Operator family	Variation	Selector
0	Central Force Dynamic	-	Direct
1	Central Force Dynamic	-	Greedy
2	Differential Mutation	/current-to-best/	Greedy
3	Differential Mutation	rand-to-best-and-current	Direct
4	Firefly Dynamic	Uniform distribution	Direct
5	Firefly Dynamic	Uniform distribution	Greedy
6	Genetic Uniform Crossover	Rank Weighting Pairing	Greedy
7	Genetic Uniform Crossover	Rank Weighting Pairing	Direct
8	Genetic Mutation	Uniform distribution	Direct
9	Genetic Mutation	Uniform distribution	Greedy
10	Gravitational Search	-	Direct
11	Gravitational Search	-	Greedy
12	Random Flight	Lévy distribution	Direct
13	Random Flight	Lévy distribution	Greedy
14	Local Random Walk	Normal distribution	Direct
15	Local Random Walk	Normal distribution	Greedy
16	Random Sample	-	Direct
17	Random Sample	-	Greedy
18	Random Search	Uniform distribution	Greedy
19	Random Search	Uniform distribution	Direct
20	Spiral Dynamic	-	Direct
21	Spiral Dynamic	-	Greedy
22	Inertial Swarm Dynamic	Uniform distribution	Greedy
23	Constricted Swarm Dynamic	Uniform distribution	Greedy

iterations. Also, a maximum of 10 HH steps and 20 replicates per candidate MH were set to provide the optimal solver. In this stage, we are not only looking for the optimal MH to tune the AVR system's PID controller but also to find which family of operators has the highest frequency for obtaining the customized MH. The main objective is clear: to deliver to the scientific community which search operators are the ones that can offer the best performance in this kind of optimization problem. The results of these experiments are presented in the following section.

V. EXPERIMENTAL RESULTS

The first stage of the experiment involved running a HH procedure 61 times using the CUSTOMHyS framework, i.e., 61 MHs were obtained for the PID controller tuning problem for the AVR system. All the results obtained in these tests can be found at https://github.com/Danielfz14/Paper_1570993779_CEC_PID_AVR. This stage was focused on monitoring the occurrence of the different heuristic operators to verify possible patterns or trends in their selection. An MH cardinality range was established for the operators to configure each customized MH, with a minimum of two and a maximum of three search operators per MH candidate. The results are presented in TABLE IV, where the Swarm Dynamic operator family was the most frequent, with 34 times (18.57%). The second most frequent operator family was Spiral Dynamic, with 22 times (12.02%). Differential Mutation and Random Flight were also significant, with 18 and 12 occurrences representing 9.83% and 6.55%, respectively. Central Force Dynamic stands out within the family of operators with lower frequency, which appeared thrice (1.63%).

TABLE IV
FREQUENCY OF SEARCH OPERATOR FAMILY USAGE.

Operator family	SO_1	SO_2	SO_3	Total
Random Search	3	2	2	7
Genetic Uniform Crossover	4	4	1	9
Differential Mutation	3	10	5	18
Random Flight	7	4	1	12
Swarm Dynamic	12	17	5	34
Spiral Dynamic	12	7	3	22
Local Random Walk	3	3	4	10
Firefly Dynamic	4	5	2	11
Genetic Mutation	5	2	1	8
Random Sample	5	3	0	8
Gravitational Search	2	3	2	7
Central Force Dynamic	1	1	1	3
None	-	-	34	34

In addition, it is essential to analyze the overall performance of this test. In Fig. 4, we can observe a box plot representing the distribution of the performance (Eq 7) of the HH process in the 61 runs. The fitness values' median is at 1.0765 (Q_2), and the mean is at 1.4194. The maximum and minimum values are 2.455 and 0.145, respectively. The first quartile (Q_1) is 0.654, while the third quartile (Q_3) is 1.416, with an interquartile range of 0.761. Some outliers are presented, highlighting the largest with a value of 9.325. Such a performance value was produced by an MH composed of the operators $h_o = h_0 \circ h_{14}$. These operators belong to the Central Force Dynamic and Local Random Walk family, where the first operator is the least crowded in the test performed.

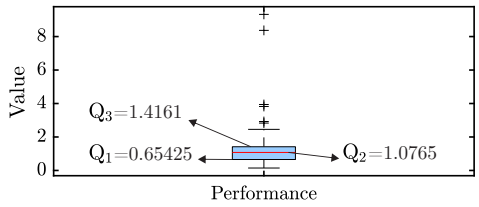

Fig. 4. Fitness values achieved by the hyper-heuristic process across 61 runs and the corresponding first to third quartiles.

Now, examining each operator in detail, Fig. 5 shows the trend presented by the first operator. In this sense, the operators with the highest appearance percentage are the Spiral Dynamic and Swarm Dynamic perturbators, each with 19.7% of the first operator's total selections. These results underline the need to start with a strategy that can effectively explore the search space from the beginning of the optimization process. This behavior continues to be observed in those perturbators that did not end up being the most selected but still presented a significant percentage of occurrence. These perturbators were: Random Flight (11.5%), Random Sample (8.2%), and Genetic Mutation (8.2%).

In Fig. 6, the selection trend for the second operator in the custom metaheuristic sequence is observed. On the one hand, the concurrence of the Swarm Dynamic perturbation is remarkable, with 27.9% concurrence. On the other hand, the Differential Mutation perturbator also

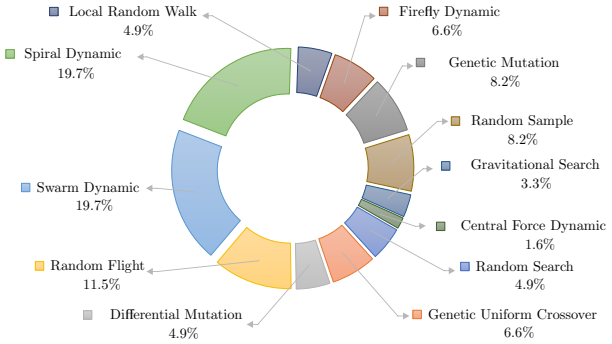

Fig. 5. The trend of operator one from the tailored MH.

shows a high concurrency percentage of 16.4%, suggesting that this operator's diversification and exploration features are of great importance in the intermediate stages of the optimization process. In some cases where only two operators or perturbators were generated, the second stage is associated with exploiting the candidate solutions.

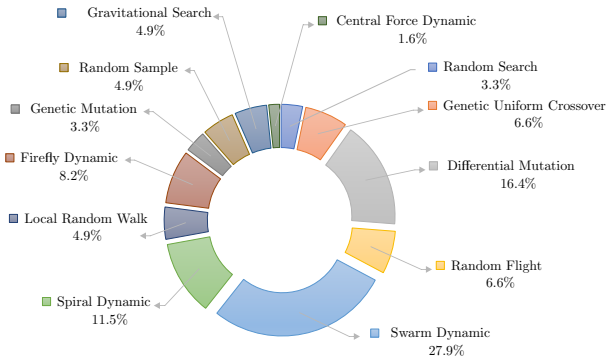

Fig. 6. The trend of operator two from the tailored MH.

Fig. 7 exhibits the frequency distribution for selecting the third operator in constructing custom metaheuristics. We observe that Swarm Dynamic and Differential Mutation are the perturbators with the highest frequency of concurrence, with 18.5% each. This selection reflects the need to include strategies exploiting previously explored candidate solutions. These perturbators are recognized for their ability to fine-tune and precisely tweak the solution space, balancing exploration of new areas with intensification around candidate solutions.

When looking at the trends obtained for each SO position together, it becomes clear that there is a trend toward using operators that include intensive exploration strategies and balanced exploitation. Swarm Dynamic, focusing on swarm intelligence and search space exploration, is a consistent choice at different stages, which tells us that its application provides remarkable benefits throughout the optimization process. Differential Mutation, recognized for its ability to introduce significant variations and avoid local minima, emerges as a critical component for refining solutions in the final stage of the MH sequence, adjusting its value in the final phase, i.e., in the search intensification.

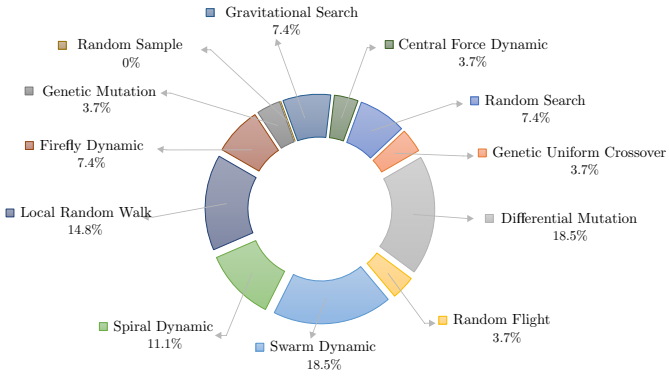

Fig. 7. The trend of operator three from the tailored MH.

Analyzing the data obtained in this test and inspecting the best three MHs and the worst, we can observe the results obtained by each configuration in the HH process (TABLE V). The configurations formed by the perturbators, namely, spiral dynamic, differential mutation, and swarm dynamic, showed the best performance. It indicates that it is not by chance that these SOs are the most used during the replications of the HH process (see Fig. 5, 6, 7). Moreover, these configurations managed to minimize the objective function to around 0.11, with a low dispersion of the results as indicated by their interquartile range value. Plus, the worst three configurations were those where no combination of these operators above was present. These MHs presented a high dispersion, which shows us a less consistent performance. This analysis emphasizes the importance of correctly selecting a sequence of operators to obtain good performances.

TABLE V
COMPARISON OF THE THREE BEST AND WORST MH CONFIGURATIONS OBTAINED DURING THE HH PROCESS REPLICATES IN TERMS OF PERFORMANCE.

SO_1	SO_2	SO_3	Best F_{obj}	Avg. ± St. Dev.	IQR	Perf. (7)
h_{20}	h_2	h_{23}	0.111	0.148±0.085	0.027	0.1450
h_{13}	h_{23}	h_2	0.114	0.154±0.085	0.016	0.1457
h_{11}	h_{23}	h_2	0.114	0.138±0.026	0.021	0.1488
h_{21}	h_{21}	None	0.646	2.368±1.560	2.040	3.941
h_6	h_3	h_0	0.754	4.475±2.430	4.211	8.369
h_0	h_{14}	h_{23}	1.105	5.254±3.417	4.440	9.325

Examining only the best MH (MH_*) obtained in the previous tests, we notice it is constituted by the operators:

- h_{20}–*Perturbator*: Spiral Dynamic with radius varying uniformly between 0.8 and 1; *Selector*: Direct.
- h_2–*Perturbator*: Differential Mutation with mutation scheme rand-to-best-and-current, and scale factor of 1.0; *Selector*: Greedy.
- h_{23}–*Perturbator*: Constriction Swarm Dynamic with a uniform distribution for sampling the random variables; *Selector*: Greedy.

To obtain this MH_*, it was necessary to perform an HH

process, as seen in Figure 8, where each box represents the fitness results obtained after repeating the current candidate MH 20 times. In the seventh step, HH, we observe that $MH_7 = MH_*$ is an algorithm that outperforms the previous candidates. Moreover, Figure 8 shows the evolution of the performance metric Q (orange dashed lines and green dots), done at step seven, the minimal value of Q obtained.

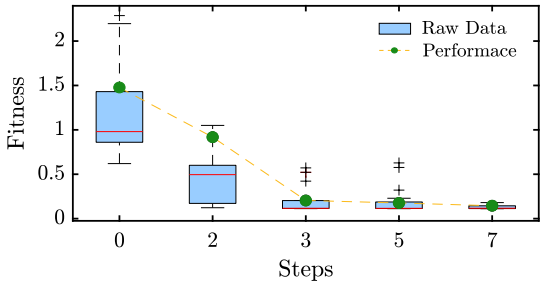

Fig. 8. Evolution trend of the low-level fitness value along the HH procedure for the best-achieved MH.

Studying the behavior of the obtained MH_7, we can observe in Figure 9 how the MH_7 managed to find a solution with a fitness value close to zero, with a high degree of repeatability and fast convergence, which are desired characteristics for this type of solvers.

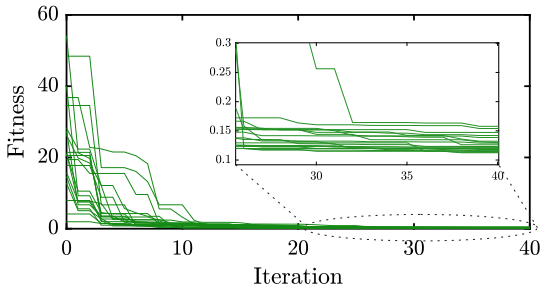

Fig. 9. Replicate testing of the tailored MH_7 for dealing with the low-level problem.

Finally, we analyze the low-level problem (the tuning of the PID controller to control the AVR system); we observe in Table VI the controller's performance-tuned by the tailored MH_* versus the performance of four MH algorithms that have published their results. These algorithms are the Tree Seed Algorithm (TSA) [22], the Improved Kidney Algorithm (IKA) [21], Particle Swarm Optimization (PSO) [38], and Differential Evolution (DE) [38]. The overshoot rendered by the MH_*-tuned controller was 1.9%, which represents 7.89, 8.19, 15.74, and 17.23 times lower than those obtained by the IKA, TSA, PSO, and DE-tuned controllers. In addition, the controller tuned with MH_* achieved a settling time of 0.454 s, which positions it as the fastest controller in this comparative analysis. In the case of steady-state error, all algorithms improved this metric, but the controller tuned with MH_* obtained the lowest value among the others. Fig. 10 summarizes the above analysis, showing the evolution of the output voltage from the AVR system for each controller adjusted via the five different algorithms.

TABLE VI
RESULTS OF TRANSIENT RESPONSE ANALYSIS OF THE AVR SYSTEM FOR DIFFERENT CONTROLLERS TUNED BY MH_*, IKA, TSA, PSO, AND DE.

Alg.	**Fobj.**	M_p **[%]**	T_s **[s]**	T_r **[s]**	E_{ss} **[p.u.]**	K_p	K_i	K_d
MH_*	Proposed	**1.96**	**0.454**	0.30	$\mathbf{7.84}\times10^{-6}$	0.643	0.438	0.205
IKA [21]	ZGL+ITSE	15.00	0.753	**0.12**	2.08×10^{-4}	1.128	0.956	0.567
TSA [22]	ITSE	15.57	0.758	0.13	8.86×10^{-4}	1.042	1.009	0.599
PSO [38]	ITSE	29.92	3.399	0.163	8.10×10^{-3}	1.777	0.382	0.318
DE [38]	ITSE	32.74	2.650	0.154	6.56×10^{-3}	1.949	0.443	0.342

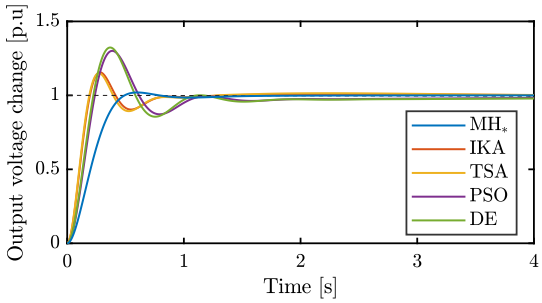

Fig. 10. Step response of the AVR system in a closed-loop and with a PID controller tuned by MH_*, TSA, IKA, PSO, and DE.

VI. CONCLUSIONS

This study demonstrated the efficacy of AAD methodologies in practical optimization scenarios, particularly in tuning a PID controller within an AVR system. By employing an HH process powered by Simulated Annealing, 61 population MHs specifically tailored to this challenge were developed. The analysis of these solutions showed a distinct preference for Spiral Dynamics, Swarm Dynamics, and Differential Mutation operators, indicating their effectiveness in this context. The sequence in which these operators are applied proved critical for optimal performance, with the combination $h_o = h_{20} \circ h_2 \circ h_{23}$ (Spiral Dynamics, Swarm Dynamics, and Differential Mutation, respectively) demonstrating superior results.

Our finding offers a valuable resource for professionals in electrical and control systems, guiding the selection or design of MHs. (Naturally, this methodology can be extended to other real-world engineering scenarios.) It bypasses the need for extensive literature review, often cluttered with 'novel' yet essentially classic MHs under different metaphors. Applying AAD streamlines the development process, minimizing human intervention in designing and tuning MH algorithms. Moreover, the impressive performance of the customized MHs in our low-level problem highlights the importance of precise hyper-parameter tuning.

In future work, we aim to leverage this trend in operators as a foundation for developing an AAD-based methodology. This approach would focus on the most effective operators, avoiding the need to explore the entire search space presented in this

study. Such a strategy promises to enhance the efficiency and precision of solving similar problems in future applications.

ACKNOWLEDGMENTS

This work was supported by the Research Group in Advanced Artificial Intelligence at Tecnológico de Monterrey, financially funded by Tecnológico de Monterrey under grants NUA A00836756, FAP 2570, and OPEX GIs 23-24, and by the Mexican Council of Humanities, Sciences, and Technologies (CONAHCyT) under scholarship 1046000.

REFERENCES

[1] X.-S. Yang, "Optimization and metaheuristic algorithms in engineering," *Metaheuristics in water, geotechnical and transport engineering*, vol. 1, p. 23, 2013.

[2] O. Bozorg-Haddad, M. Solgi, and H. A. Loáiciga, *Meta-heuristic and evolutionary algorithms for engineering optimization*. John Wiley & Sons, 2017.

[3] K. Rajwar, K. Deep, and S. Das, "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges," *Artificial Intelligence Review*, pp. 1–71, 2023.

[4] H. Su, D. Zhao, A. A. Heidari, L. Liu, X. Zhang, M. Mafarja, and H. Chen, "Rime: A physics-based optimization," *Neurocomputing*, vol. 532, pp. 183–214, 2023.

[5] O. Kramer and O. Kramer, *Genetic algorithms*. Springer, 2017.

[6] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical science*, vol. 8, no. 1, pp. 10–15, 1993.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization (PSO)," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[8] F. Campelo and C. Aranha, "Sharks, zombies and volleyball: Lessons from the evolutionary computation bestiary," in *LIFELIKE Computing Systems Workshop 2021*. CEUR-WS. org, 2021.

[9] J. de Armas, E. Lalla-Ruiz, S. L. Tilahun, and S. Voß, "Similarity in metaheuristics: a gentle step towards a comparison methodology," *Natural Computing*, vol. 21, no. 2, pp. 265–287, 2022.

[10] A. Tzanetos and G. Dounias, "Nature inspired optimization algorithms or simply variations of metaheuristics?" *Artificial Intelligence Review*, vol. 54, pp. 1841–1862, 2021.

[11] A. P. Piotrowski, J. J. Napiorkowski, and P. M. Rowinski, "How novel is the "novel" black hole optimization approach?" *Information Sciences*, vol. 267, pp. 191–200, 2014.

[12] J. Cruz-Duarte, J. Ortiz-Bayliss, I. Amaya, Y. Shi, H. Terashima-Marín, and N. Pillay, "Towards a Generalised Metaheuristic Model for Continuous Optimisation Problems," *Mathematics*, vol. 8, no. 11, p. 2046, nov 2020.

[13] H. Stegherr and J. Hähner, "Analysing metaheuristic components," in *LIFELIKE*, 2021.

[14] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and Y. Shi, "Hyper-heuristics to customise metaheuristics for continuous optimisation," *Swarm and Evolutionary Computation*, vol. 66, p. 100935, 2021.

[15] M. A. L. Silva, S. R. de Souza, M. J. F. Souza, and M. F. de Franca Filho, "Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis," *Applied Soft Computing*, vol. 71, pp. 433–459, 2018.

[16] F. Da Ros and L. Di Gaspero, "Exploring the potential of jules: A white box framework for local search metaheuristics," in *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal*. ACM, New York, NY, USA, 2023, p. 4.

[17] D. F. Zambrano-Gutierrez, J. M. Cruz-Duarte, J. G. Avina-Cervantes, J. C. Ortiz-Bayliss, J. J. Yanez-Borjas, and I. Amaya, "Automatic design of metaheuristics for practical engineering applications," *IEEE Access*, vol. 11, pp. 7262–7276, 2023.

[18] D. F. Zambrano-Gutierrez, J. Cruz-Duarte, and H. Castañeda, "Automatic hyper-heuristic to generate heuristic-based adaptive sliding mode controller tuners for buck-boost converters," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 1482–1489.

[19] A. Nikolikj, R. Lang, P. Korošec, and T. Eftimov, "Explaining differential evolution performance through problem landscape characteristics," in *International Conference on Bioinspired Optimization Methods and Their Applications*. Springer, 2022, pp. 99–113.

[20] T. Li, W. Bai, Q. Liu, Y. Long, and C. P. Chen, "Distributed fault-tolerant containment control protocols for the discrete-time multiagent systems via reinforcement learning method," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[21] S. Ekinci and B. Hekimoğlu, "Improved kidney-inspired algorithm approach for tuning of pid controller in avr system," *IEEE Access*, vol. 7, pp. 39 935–39 947, 2019.

[22] E. Köse, "Optimal control of avr system with tree seed algorithm-based pid controller," *IEEE Access*, vol. 8, pp. 89 457–89 467, 2020.

[23] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, H. Terashima-Marín, and Y. Shi, "CUSTOMHyS: Customising Optimisation Metaheuristics via Hyper-heuristic Search," *SoftwareX*, vol. 12, p. 100628, Jul. 2020.

[24] S. S. Rao, *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[25] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, *A Classification of Hyper-Heuristic Approaches: Revisited*. Springer, 2019, pp. 453–477.

[26] D. B. Lenat, "The nature of heuristics," *Artificial intelligence*, vol. 19, no. 2, pp. 189–249, 1982.

[27] J. Cruz-Duarte, J. Ortiz-Bayliss, I. Amaya, and N. Pillay, "Global optimisation through hyper-heuristics: Unfolding population-based metaheuristics," *Applied Sciences*, vol. 11, no. 12, p. 5620, jun 2021.

[28] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications*. Springer, 2018.

[29] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[30] R. Qu, G. Kendall, and N. Pillay, "The general combinatorial optimisation problem: Towards automated algorithm design," *IEEE Comput. Intell. Mag.*, vol. 15, no. 2, pp. 14–23, 2020.

[31] Q. Zhao, B. Yan, T. Hu, X. Chen, and Y. Shi, "Autooptlib: A library of automatically designing metaheuristic optimization algorithms in matlab," *arXiv preprint arXiv:2303.06536*, 2023.

[32] J. Cruz-Duarte, I. Amaya, J. Ortiz-Bayliss, S. Conant-Pablos, H. Terashima-Marín, and Y. Shi, "Hyper-heuristics to customise metaheuristics for continuous optimisation," *Swarm Evol. Comput.*, vol. 66, p. 100935, 2021.

[33] C. Audet and D. Orban, "Finding optimal algorithmic parameters using derivative-free optimization," *SIAM Journal on Optimization*, vol. 17, no. 3, pp. 642–664, 2006.

[34] P. M. Patil and S. Patil, "Automatic voltage regulator," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE, 2020, pp. 1–5.

[35] A. Vinogradov, A. Vinogradova, I. Golikov, and V. Bolshev, "Adaptive automatic voltage regulation in rural 0.38 kv electrical networks," *International Journal of Emerging Electric Power Systems*, vol. 20, no. 3, p. 20180269, 2019.

[36] G. A. Salman, A. S. Jafar, and A. I. Ismael, "Application of artificial intelligence techniques for lfc and avr systems using pid controller," *International Journal of Power Electronics and Drive Systems*, vol. 10, no. 3, p. 1694, 2019.

[37] M. A. Sahib and B. S. Ahmed, "A new multiobjective performance criterion used in pid tuning optimization algorithms," *Journal of advanced research*, vol. 7, no. 1, pp. 125–134, 2016.

[38] H. Gozde and M. C. Taplamacioglu, "Comparative performance analysis of artificial bee colony algorithm for automatic voltage regulator (avr) system," *Journal of the Franklin Institute*, vol. 348, no. 8, pp. 1927–1946, 2011.